Defence Research and Development Laboratory

Internship Report

May 7th 2024 – June 7th 2024

OBJECT DETECTION USING YOLOV7

In partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE ENGINEERING

Vasamsetti Akanksha Deepthi 1608-21-733-146

Submitted to
Dr. D Penchalaiah
Sc-'F',DRDL

Defence Research and Development Laboratory
DRDO – Ministry of Defence
Kanchanbagh, Hyderabad-500058

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to Shri. Penchaliah, Scientist 'F', Shri Vaddi Chandra Shekar, Scientist 'C', Shri. Boda Nehru, Scientist 'C' and Shri G.A. Srinivas Murthy, Director of DRDL, Defence Research & Development Organization, for granting me the invaluable opportunity to gain work experience in this esteemed organization and acquire knowledge in the field of Artificial Intelligence and Neural Networks.

I extend my sincere appreciation to Shri. Penchaliah, Scientist 'F', for offering me the internship opportunity and providing valuable guidance throughout this endeavor. Furthermore, I would like to convey my deep appreciation to Shri. Vaddi Chandra Shekar, Scientist 'C', and Shri. Boda Nehru, Scientist 'C' of DRDL, for their invaluable mentorship during this project.

INDUSTRIAL TRAINING

Introduction
Artificial Intelligence (AI) and Machine Learning (ML) are transformative technologies that have revolutionized numerous industries and are driving innovation in today's digital age. This document provides a concise overview of AI and ML, explaining their fundamental concepts and applications.

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and learn like humans. It involves the development of computer systems capable of performing tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and problem-solving.

AI can be categorized into two types: Narrow AI and General AI. Narrow AI, also known as Weak AI, is designed to perform specific tasks and excel in narrow domains. General AI, on the other hand, aims to exhibit human-like intelligence across a wide range of tasks and domains.

AI encompasses various subfields, including Natural Language Processing (NLP), Computer Vision, Robotics, Expert Systems, and Neural Networks.
These subfields utilize different techniques and algorithms to achieve specific AI functionalities.

Machine Learning (ML): is a subset of AI that focuses on the development of algorithms and statistical models that enable computer systems to learn from data and make predictions or decisions without being explicitly programmed. ML algorithms learn patterns and relationships from the data and use that knowledge to perform tasks or make predictions.
ML can be categorized into three types: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Supervised Learning involves training ML models using labelled data.

Unsupervised Learning, on the other hand, deals with unlabelled data and aims to discover hidden patterns or structures within the data. Reinforcement Learning involves training models through an interactive process of trial and error, where the model learns from feedback or rewards received based on its actions. ML

algorithms include Decision Trees, Support Vector Machines, Naive Bayes, Neural Networks, and more. These algorithms are used for various tasks, such as classification, regression, clustering, and recommendation systems.

Applications of AI and ML:

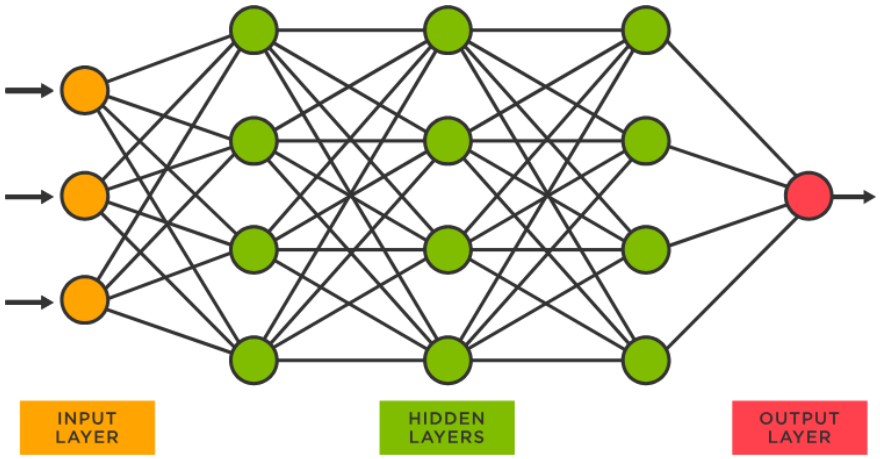
AI and ML have found applications across numerous industries, transforming the way businesses operate and improving various aspects of our lives. Some key applications include:

1. Healthcare: AI and ML are used for disease diagnosis, drug discovery, personalized medicine, patient monitoring, and healthcare management systems.
2. Finance: AI and ML algorithms are used for fraud detection, risk assessment, algorithmic trading, credit scoring, and customer service automation.
3. Transportation: AI and ML technologies enable autonomous vehicles, traffic optimization, predictive maintenance, and route planning.
4. Retail: AI and ML are used for personalized marketing, demand
5. forecasting, inventory management, and recommendation systems.
6. Education: AI and ML are applied in adaptive learning systems, intelligent tutoring, plagiarism detection, and educational data analysis.
7. Manufacturing: AI and ML are used for predictive maintenance, quality control, supply chain optimization, and robotics automation.
8. Defense: AI in defense enables autonomous systems for surveillance and combat, enhances threat detection and analysis capabilities, and strengthens cybersecurity measures against evolving cyber threats. It also provides decision support systems and aids in intelligence gathering and analysis for enhanced situational awareness

A neural network is an interconnected assembly of simple processing elements units or nodes whose functionality is loosely based on animal neurons. In simple words, the mechanism of neural networks mimics that of a human or animal brain. The processing ability of the network will be stored in the inter-unit connection strengths, or weights, obtained by a process of adaptation to or learning from a set of training patterns.

Neural networks and deep learning are big topics in Computer Science and in the technology industry, they currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. Recently many papers have been published featuring AI that can learn to paint, build 3D Models, and create user interfaces(pix2code), some create images given

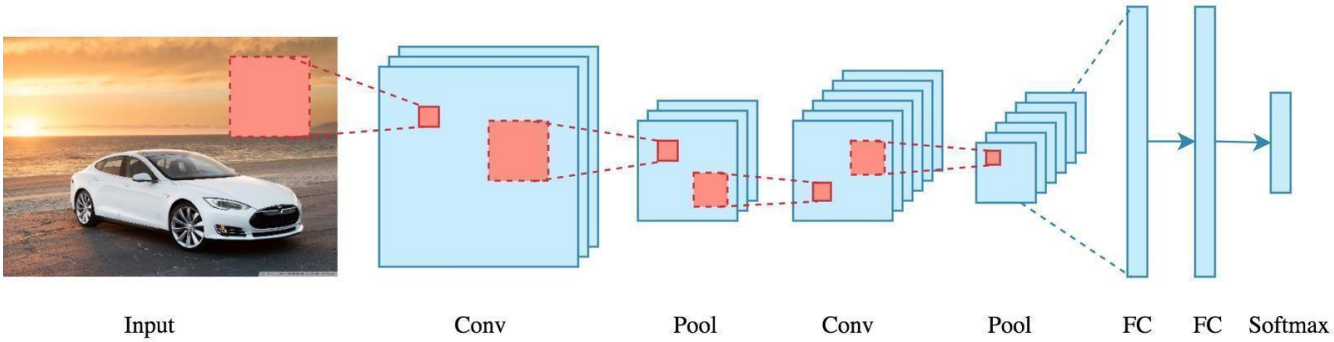
a sentence and there are many more incredible things being done every day using neural networks.



It is this architecture and style of processing of a biological neuron that we hope to incorporate in neural networks and, because of the emphasis on the importance of the inter-neuron connections, this type of system is sometimes referred to as being connectionist and the study of this general approach as connectionism. This terminology is often the one encountered for neural networks in the context of psychologically inspired models of human cognitive function. However, we will use it quite generally to refer to neural networks without reference to any particular field of application.

Convolutional Neural Networks

In deep learning, a Convolutional Neural Network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolutional. Now in mathematics convolutional is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.



a) Inside Convolutional neural networks

CNN is another type of neural network that can uncover key information in both time series and image data. For this reason, it is highly valuable for image-related tasks, such as image recognition, object classification and pattern recognition. To identify patterns within an image, a CNN leverages principles from linear algebra, such as matrix multiplication. CNNs can also classify audio and signal data.

A CNN's architecture is analogous to the connectivity pattern of the human brain. Just like the brain consists of billions of neurons, CNNs also have neurons arranged in a specific way. In fact, a CNN's neurons are arranged like the brain's frontal lobe, the area responsible for processing visual stimuli. This arrangement ensures that the entire visual field is covered, thus avoiding the piecemeal image processing problem of traditional neural networks, which must be fed images in reduced-resolution pieces. Compared to the older networks, a CNN delivers better performance with image inputs, and also with speech or audio signal inputs.

b) CNN layers

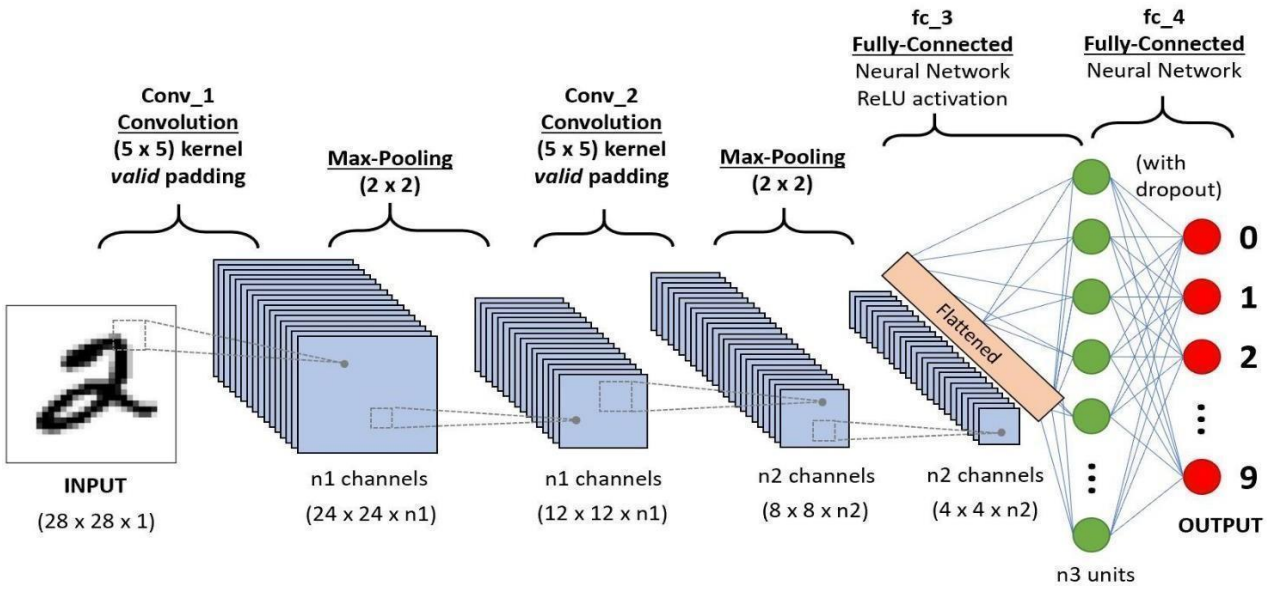
A deep-learning CNN consists of three layers: a convolutional layer, a pooling layer, and a fully connected (FC) layer. The convolutional layer is the first layer while the FC layer is the last. From the convolutional layer to the FC layer, the complexity of the CNN increases. It is this increasing complexity that allows the CNN to successfully identify larger portions and more complex features of an image until it finally identifies the object in its entirety.

Convolutional layer. The majority of computations happen in the convolutional layer, which is the core building block of a CNN. A second convolutional layer can follow the initial convolutional layer. The process of convolution involves a kernel or filter inside this layer moving across the receptive fields of the image, checking if a feature is present in the image. Over multiple iterations, the kernel sweeps over the entire image. After each iteration, a dot product is calculated between the input pixels and the filter. The final output from the series of dots is known as a feature map or convolved feature. Ultimately, the image is converted into numerical values in this layer, allowing the CNN to interpret the image and extract relevant patterns.

Pooling layer. Like the convolutional layer, the pooling layer also sweeps a kernel or filter across the input image. But unlike the convolutional layer, the pooling layer reduces the number of parameters in the input and also results in some information loss. On the positive side, this layer reduces complexity and improves the efficiency of the CNN.

Fully connected layer. The FC layer is where image classification happens in the CNN based on the features extracted in the previous layers. Here, fully connected means that all the inputs or nodes from one layer are connected to every activation unit or node of the next layer.
All the layers in the CNN are not fully connected because it would result in an unnecessarily dense network. It also would increase losses and affect the output quality, and it would be computationally expensive.

How do convolutional neural networks work?
A CNN can have multiple layers, each of which learns to detect the different features of an input image. A filter or kernel is applied to each image to produce an output that gets progressively better and more detailed after each layer. In the lower layers, the filters can start as simple features.
At each successive layer, the filters increase in complexity to check and identify features that uniquely represent the input object. Thus, the output of each convolved image the partially recognized image after each layer -- becomes the input for the next layer. In the last layer, which is an FC layer, the CNN recognizes the image or the object it represents.
With convolution, the input image goes through a set of these filters. As each filter activates certain features from the image, it does its work and passes on its output to the filter in the next layer. Finally, all the image data progressing through the CNN's multiple layers allow the CNN to identify the entire object.



CNNs vs. Neural Networks

The biggest problem with regular neural networks (NNs) is a lack of scalability. For smaller images with fewer color channels, a regular NN may produce satisfactory results. But as the size and complexity of an image increases, the need for computational power and resources also increases which necessitates a larger and more expensive NN.

Moreover, the problem of overfitting also arises over time, wherein the NN tries to learn too many details in the training data. It may also end up learning the noise in the data, which affects its performance on test data sets. Ultimately, the NN fails to identify the features or patterns in the data set and thus the object itself.

Object Detection

Object detection is a computer vision task that involves identifying and locating objects within an image or video frame. The goal is to accurately recognize various objects and determine their precise locations by drawing bounding boxes around them. This process typically involves multiple steps, including feature extraction, object classification, and bounding box regression.

In simpler terms, imagine looking at a photo and being able to automatically identify where different objects are, like cars, people, or animals, and drawing boxes around them to outline their positions. This capability is crucial in numerous applications, such as surveillance, autonomous vehicles, and image retrieval. Object detection algorithms utilize various techniques, including traditional machine learning approaches like Support Vector Machines (SVMs) and more modern deep learning methods like Convolutional Neural Networks (CNNs). Deep learning models, particularly CNNs, have significantly advanced the accuracy and efficiency of object detection systems in recent years.

Overall, object detection plays a vital role in enabling machines to understand and interact with visual data, paving the way for numerous applications in fields such as robotics, healthcare, and security.

How does Object Detection Work?
In general, object detection works by analyzing an input image or video frame and identifying the presence and location of objects within it. Here's a simplified overview of the process:

1. Input Image: The object detection process begins with an input image or video frame. This image is typically represented as a grid of pixels, with each pixel containing color information.

2. Feature Extraction: Next, the object detection algorithm extracts meaningful features from the input image. These features may include edges, textures, shapes, or other visual patterns that can help distinguish different objects.

3. Object Localization: Once features are extracted, the algorithm scans the image to identify regions that may contain objects. This step is often referred to as object localization. Techniques such as sliding window or region proposal algorithms are commonly used to generate candidate regions where objects might be present.

4. Object Classification: For each candidate region, the algorithm performs object classification to determine the type or category of object present (e.g., car, person, dog). This step involves feeding the extracted features into a classifier, such as a Support Vector Machine (SVM) or a Convolutional Neural Network (CNN), trained to recognize specific object classes.

5. Bounding Box Regression: In addition to classifying objects, the algorithm also refines the location of detected objects by predicting bounding boxes that tightly enclose them. This step is known as bounding box regression and involves adjusting the position and size of candidate regions to better fit the detected objects.

6. Post-processing: Finally, post-processing techniques may be applied to filter out false positives, refine object boundaries, and improve the overall accuracy of the detection results. Non-maximum suppression is a common post-processing technique used to remove overlapping bounding boxes and retain only the most confident detections.

By combining these steps, object detection algorithms can effectively identify and locate objects within images or video frames, enabling a wide range of applications in fields such as autonomous driving, surveillance, augmented reality, and more.

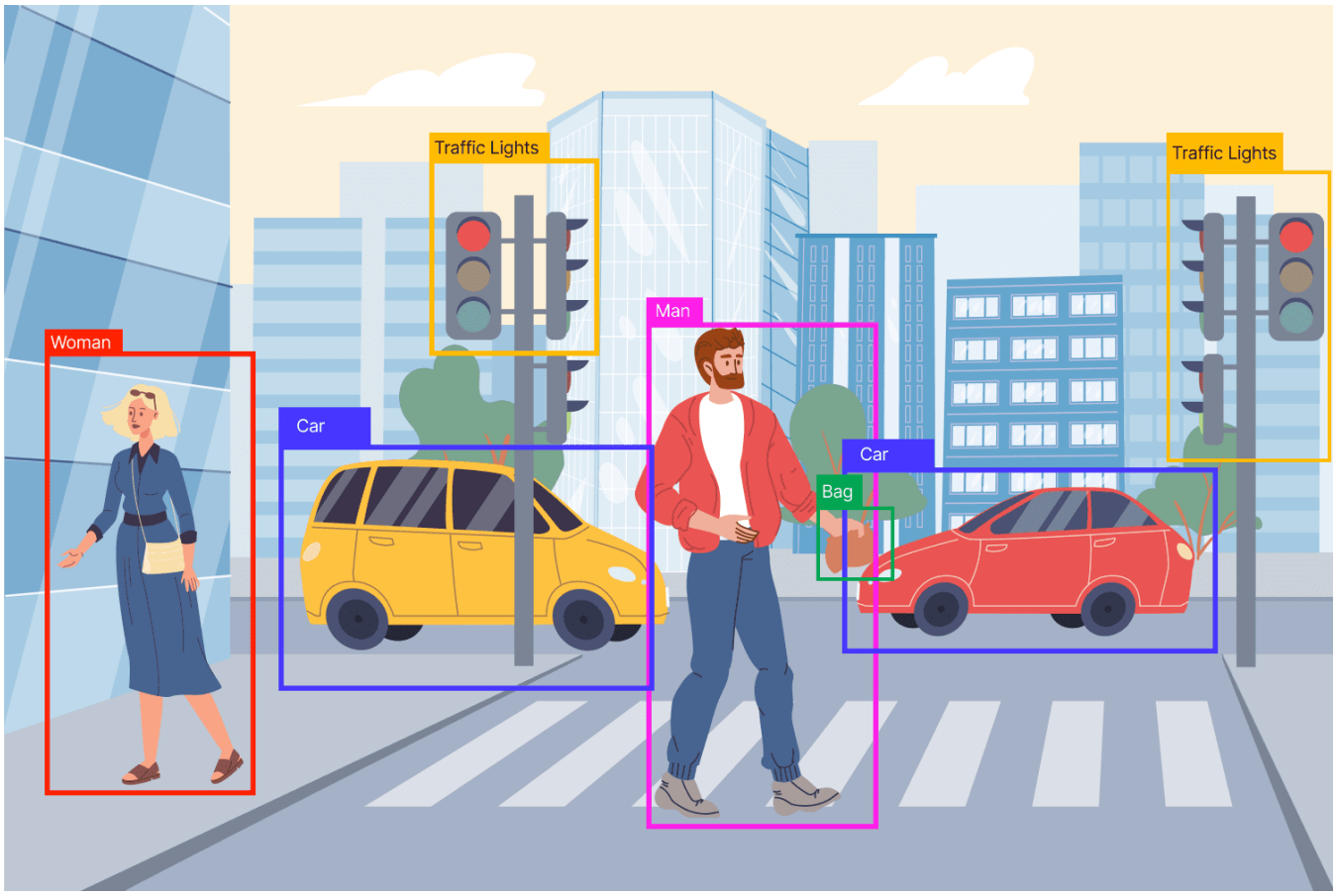


Figure. Object Detection

Technologies Available for Object Detection

Object detection is a crucial task in computer vision, enabling machines to identify and locate objects within images or videos. Over the years, various technologies have emerged to address object detection challenges, ranging from traditional methods to modern deep learning techniques. Here's an overview of some of the key technologies available for object detection:

1. **Traditional Computer Vision Techniques**:
   - **Haar Cascades**: Developed by Viola and Jones, Haar cascades are used for object detection by analyzing specific features in an image. It's effective for detecting faces and other simple objects but less accurate for complex scenarios.
   - **Histogram of Oriented Gradients (HOG):** HOG extracts features based on local intensity gradients and is commonly used for pedestrian detection and similar tasks.

- **SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features)**: These algorithms are robust to scaling, rotation, and lighting changes, making them suitable for object recognition tasks.

2. Machine Learning Approaches:

- **Support Vector Machines (SVM):** SVM classifiers are widely used for object detection when combined with appropriate feature extraction techniques. They work well with high-dimensional data and can handle non-linear decision boundaries.
- **Random Forests and Decision Trees:** These ensemble learning methods are effective for both classification and regression tasks, including object detection.
- **Template Matching:** This technique involves comparing a template image with different parts of the input image to find the best match. While simple, it's computationally expensive and sensitive to variations in scale and orientation.

3. Deep Learning-based Approaches:

- **Convolutional Neural Networks (CNNs):** CNNs have revolutionized object detection by automatically learning features from raw pixel data. Architectures like AlexNet, VGG, ResNet, and Inception have been adapted for object detection tasks.
- **Region-based Convolutional Neural Networks (R-CNN):** R-CNN and its variants (Fast R-CNN, Faster R-CNN, Mask R-CNN) propose regions of interest in an image and then classify and refine these regions. They achieve high accuracy but are relatively slow.
- **You Only Look Once (YOLO):** YOLO models are designed for real-time object detection by directly predicting bounding boxes and class probabilities from the entire image in a single pass. YOLOv3 and YOLOv4 are popular versions known for their speed and accuracy.
- **Single Shot MultiBox Detector (SSD):** SSD is another real-time object detection algorithm that combines high accuracy with fast inference speed. It predicts multiple bounding boxes and class probabilities at different scales within a single network.
- **EfficientDet:** This model architecture improves upon prior detectors by efficiently scaling the network's depth and width to achieve better accuracy and speed trade-offs across various resource constraints.

4. Transfer Learning and Pre-trained Models:

- Pre-trained models such as those from the TensorFlow Object Detection API, PyTorch torchvision library, or models trained on large datasets like COCO or ImageNet serve as starting points for fine-tuning on specific object detection tasks.

Transfer learning reduces the need for large annotated datasets and accelerates model training.

5. Hardware Acceleration:

- Specialized hardware like Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Field-Programmable Gate Arrays (FPGAs) significantly speed up the inference process for deep learning models, enabling real-time object detection on various platforms.

6. Advanced Techniques:

- **Attention Mechanisms:** Attention mechanisms improve object detection by allowing models to focus on relevant regions of the input image.
- **Graph Neural Networks (GNNs):** GNNs can capture relationships between objects in a scene, enhancing the contextual understanding of object detection systems.
- **3D Object Detection:** Technologies for detecting objects in three-dimensional space, such as LiDAR-based approaches combined with deep learning, are gaining importance in applications like autonomous driving and robotics.

Object Detection using YOLOv5

What does YOLOv5 do?

YOLOv5 (You Only Look Once version 5) is an object detection algorithm that accurately identifies and localizes objects within images or video frames in real-time. It is part of the YOLO (You Only Look Once) family of models, known for their speed and efficiency in object detection tasks.

How does it work?

YOLOv5 follows the principle of one-stage object detection, where it predicts bounding boxes and class probabilities directly from the entire image in a single forward pass through the neural network. This approach enables YOLOv5 to achieve real-time inference speeds while maintaining high accuracy.

Architecture of YOLOv5

The architecture of YOLOv5 consists of several key components:

1. Backbone Network: YOLOv5 uses a convolutional neural network (CNN) as its backbone to extract features from the input image. The backbone network is typically a variant of the EfficientNet architecture, known for its balance between model size and performance.

2. Neck Network: YOLOv5 incorporates a neck network that further refines the features extracted by the backbone network. This typically involves additional convolutional layers and feature fusion techniques to capture more context and spatial information.

3. Detection Head: The detection head of YOLOv5 consists of a series of convolutional layers followed by a final prediction layer. This prediction layer outputs bounding box coordinates, object class probabilities, and confidence scores for each detected object.

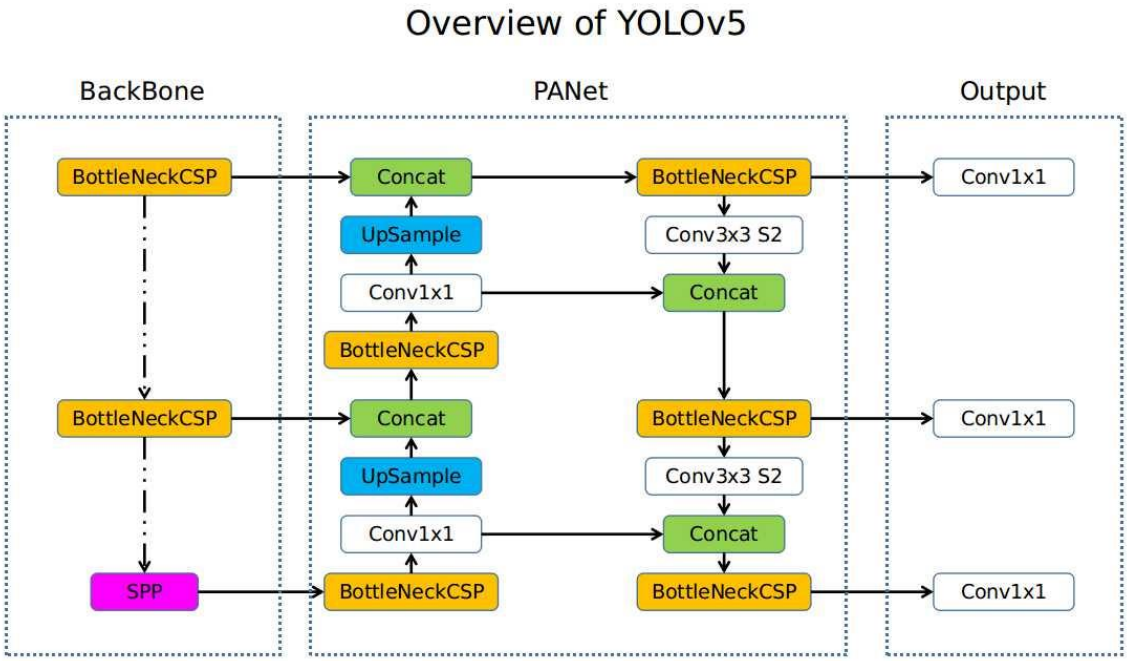


Figure. Overview of YOLOv5

How is it better than other methods?

YOLOv5 offers several advantages over previous versions and alternative object detection methods:

- Improved Accuracy: YOLOv5 achieves higher accuracy compared to earlier versions by leveraging larger and more diverse datasets for training and adopting state-of-the-art architectural advancements.

- Efficiency: YOLOv5 is designed for efficiency, allowing for real-time object detection on a wide range of hardware platforms, including CPUs, GPUs, and edge devices.

- Simplicity: YOLOv5 simplifies the object detection pipeline by performing all computations in a single pass through the neural network, resulting in faster inference speeds and reduced computational overhead.

Limitations in this Technology

While YOLOv5 offers significant improvements over previous versions and alternative methods, it still has some limitations and challenges:

- Scale Variation: YOLOv5 may struggle with accurately detecting small or highly-scaled objects, especially when they appear in cluttered scenes or have low resolution.

- Fine-Grained Classification: YOLOv5 may not perform as well on tasks requiring fine-grained object classification, where distinguishing between similar object categories is crucial.

- Limited Contextual Understanding: YOLOv5's single-pass architecture may limit its ability to capture long-range dependencies and contextual information, particularly in complex scenes with multiple objects.

- Training Data Bias: Like any deep learning model, YOLOv5 is susceptible to biases in the training data, which can lead to inaccuracies or biases in the model's predictions.

Despite these limitations, ongoing research and development efforts aim to address these challenges and further improve the performance and capabilities of YOLOv5 for a wide range of object detection applications.

In summary, YOLOv5 is a highly efficient and accurate object detection algorithm that offers real-time performance and state-of-the-art accuracy. Its streamlined architecture and innovative design make it a powerful tool for a variety of computer vision tasks, with ongoing improvements and advancements continuing to push the boundaries of object detection technology.

Object Detection using YOLOv7

What does YOLOv7 do?

YOLOv7 is a state-of-the-art object detection algorithm designed to detect and classify multiple objects within images or video frames in real-time. It builds upon the YOLO (You Only Look Once) family of object detectors, aiming to improve accuracy, speed, and efficiency over previous iterations.

How does it work?
YOLOv7 operates by dividing an input image into a grid of cells and predicting bounding boxes and class probabilities directly from these grid cells. The core idea behind YOLO is its unified model that simultaneously predicts multiple bounding boxes and their corresponding class probabilities for these boxes. This unified approach makes YOLOv7 very fast and efficient, as it avoids the need for a complex pipeline of separate components for object proposal generation and classification.

The model is trained using a single end-to-end convolutional neural network (CNN). During training, YOLOv7 learns to predict bounding boxes (coordinates of the box corners), confidence scores for these boxes (indicating the likelihood of containing an object), and class probabilities (the likelihood of the object belonging to a particular class). YOLOv7 is trained using labeled datasets and optimized to minimize prediction errors using techniques like backpropagation and gradient descent.

Architecture of YOLOv7:

The architecture of YOLOv7 is built upon a deep convolutional neural network. It typically consists of multiple convolutional layers followed by upsampling and downsampling operations, designed to extract features at different scales and resolutions from the input image. YOLOv7 often utilizes techniques like skip connections (as seen in the YOLOv3 model) to combine features from different stages of the network, aiding in more accurate localization and classification.

The key architectural components of YOLOv7 may include:
- Backbone network (like a Darknet or CSPDarknet) for feature extraction
- Neck network for feature fusion across different scales
- Detection head for predicting bounding boxes and class probabilities

YOLOv7 may adopt different backbone architectures and modifications to improve performance and efficiency.

Advantages over other methods:

YOLOv7 introduces several improvements and advantages over previous versions and alternative object detection methods:
- Enhanced accuracy: YOLOv7 achieves higher accuracy compared to earlier versions by incorporating advanced architectural modifications.
- Improved speed: YOLOv7 is optimized for faster inference speed, making it suitable for real-time applications.
- Versatility: YOLOv7 is capable of detecting a wide range of objects across different scales and orientations in images and videos.
- Simplicity: YOLOv7's unified architecture simplifies the object detection pipeline, reducing complexity and improving efficiency.

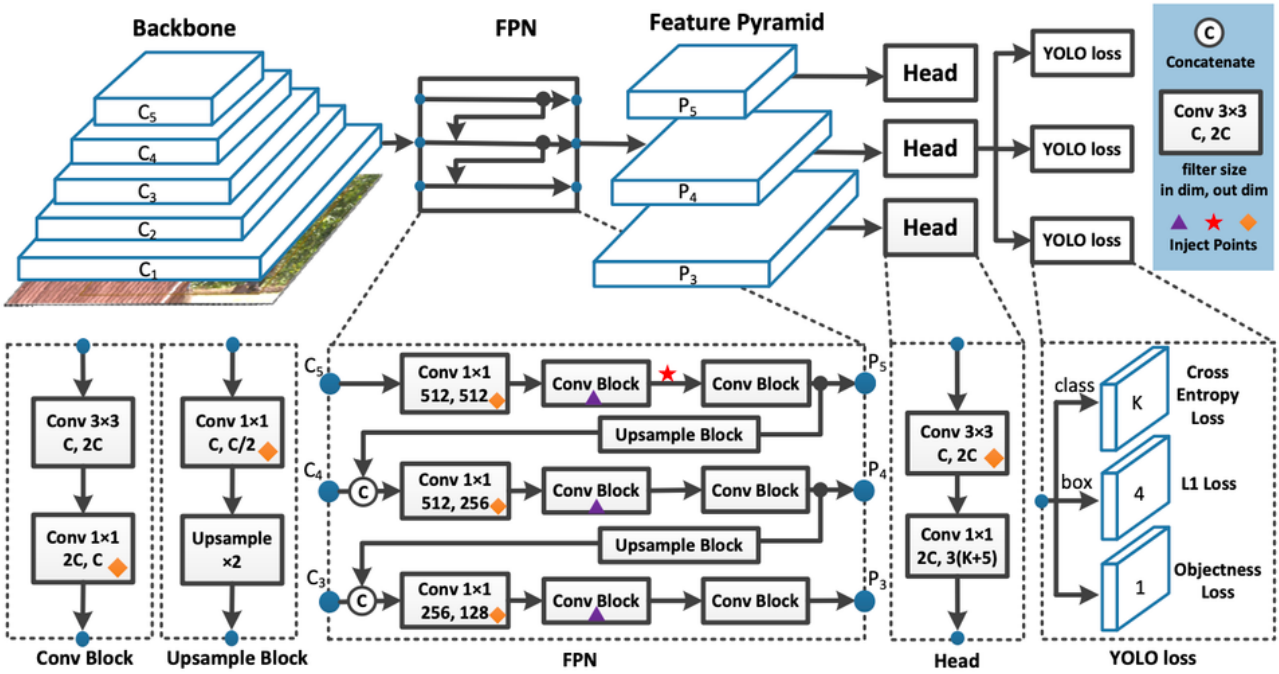


Figure. Architecture of YOLOv7

Limitations in this Technology:

Despite its strengths, YOLOv7 may have some limitations and challenges:
- Training data diversity: YOLOv7's performance heavily relies on the diversity and quality of the training dataset. Limited or biased training data can lead to decreased generalization and accuracy.

- Scale variation: While YOLOv7 is designed to handle object detection at different scales, extreme variations in object sizes within the same image can still pose challenges.
- Fine-grained object detection: YOLOv7 may struggle with detecting very small or densely packed objects due to limitations in feature resolution and receptive fields.
- Resource requirements: YOLOv7's performance may be resource-intensive during training and inference, requiring powerful hardware for optimal performance.

Addressing these limitations often involves further research and development, such as dataset curation, architectural enhancements, or specialized training strategies tailored to specific application domains.

Why Yolov7 over Yolov5 for Object Detection

Choosing YOLOv7 over YOLOv5 for object detection depends on several factors, including performance, architecture, features, and specific use cases. Here's a detailed comparison highlighting why YOLOv7 might be preferred over YOLOv5 in certain scenarios:

1. Improved Performance:

- Accuracy: YOLOv7 generally offers improved accuracy compared to YOLOv5. This improvement comes from advancements in model architecture, feature extraction, and training techniques. YOLOv7 may achieve higher mAP (mean Average Precision) scores across different datasets and object categories.
- Speed: While YOLOv5 is known for its fast inference speed, YOLOv7 can often maintain comparable speed while delivering higher accuracy. The architecture optimizations in YOLOv7 enable efficient feature extraction and object detection, making it suitable for real-time applications without compromising on performance.

2. Enhanced Architecture:

- Backbone Network: YOLOv7 typically incorporates a more advanced backbone network (such as CSPDarknet or other variants) compared to YOLOv5. This backbone enhances feature representation and extraction, leading to better object detection capabilities.
- Feature Fusion: YOLOv7 may utilize more sophisticated techniques for feature fusion across different scales and resolutions. This enables the model to effectively handle objects of varying sizes and aspect ratios within the same image.

3. Versatility and Flexibility:

- Detection Capabilities: YOLOv7 is designed to be versatile and robust in detecting a wide range of objects across different scenarios. It can handle complex scenes, occlusions, and overlapping objects more effectively than YOLOv5.
- Adaptability: YOLOv7 may be more adaptable to specific use cases and domains due to its improved architecture and feature representation. It can generalize better across diverse datasets and object categories.

4. State-of-the-Art Features:

- Incorporation of Latest Research: YOLOv7 often integrates state-of-the-art advancements in object detection research. This includes novel techniques for feature extraction, attention mechanisms, or architecture modifications that contribute to superior performance.
- Continued Development: YOLOv7 reflects ongoing development and refinement in the YOLO series, incorporating lessons learned from previous versions and addressing specific challenges faced by earlier models like YOLOv5.

5. Specific Use Cases:

- Complex Scenes: YOLOv7 excels in scenarios with complex scenes, multiple objects, and diverse object types. It can reliably detect objects even under challenging conditions.
- High-Resolution Detection: For applications requiring high-resolution object detection (e.g., surveillance systems, satellite imagery analysis), YOLOv7 may offer better performance and accuracy compared to YOLOv5.

Choosing YOLOv7 over YOLOv5 for object detection often comes down to a balance between performance, versatility, and specific application requirements. YOLOv7 is typically favored when higher accuracy, improved architecture, and enhanced detection capabilities are crucial for the task at hand. However, it's essential to consider factors such as computational resources, training data availability, and deployment constraints when making this decision.

ABSTRACT

This project presents a novel approach to real-time object detection utilizing the YOLOv7 (You Only Look Once version 7) model. The primary objective is to achieve robust and efficient object detection in images, ensuring high accuracy and fast processing times. Leveraging pre-trained weights from the COCO (Common Objects in Context) dataset, the YOLOv7 model demonstrates exceptional generalization capabilities across diverse object categories and visual contexts. This project focuses on a custom dataset comprising images of cats, dogs, and rabbits, showcasing the model's adaptability to specific use cases. Extensive evaluations on benchmark image datasets reveal the effectiveness of the YOLOv7 model in delivering accurate and real-time object detection performance. The integration of YOLOv7 offers a comprehensive solution for various applications such as surveillance, autonomous driving, and image analysis, setting a foundation for future advancements in object detection technology.

OBJECTIVE OF THE PROJECT

The project aims to achieve the following objectives:

1. **Implement Object Detection Using YOLOv7:**
    - Utilize the YOLOv7 model for accurate and efficient object detection in static images.
    - Integrate pre-trained weights from the COCO dataset to detect a wide range of objects across different categories.
2. **Develop an Efficient Image Detection Pipeline:**
    - Design a streamlined pipeline for object detection using YOLOv7, optimizing for processing efficiency and speed on image datasets.
    - Ensure compatibility with various image formats and resolutions for flexible deployment.
3. **Enhance Detection Accuracy and Generalization:**
    - Fine-tune the YOLOv7 model to improve detection accuracy on specific object categories or challenging visual contexts present in image data.
    - Enhance the generalization capabilities of the model to detect objects accurately across diverse image datasets.
4. **Evaluate Detection Performance on Image Datasets:**
    - Conduct thorough evaluation of the YOLOv7-based object detection system on benchmark image datasets and real-world image collections.
    - Measure key performance metrics such as precision, recall, and mAP (mean Average Precision) to assess the effectiveness of the detection algorithm.
5. **Optimize Model Deployment for Image Analysis:**

- Optimize the YOLOv7 model for efficient deployment on various hardware platforms and software environments for image analysis tasks.
- Explore techniques such as model quantization and optimization to reduce inference time and resource requirements.

6. Explore Practical Applications in Image Analysis:

- Explore potential applications of YOLOv7-based object detection in image analysis tasks such as medical imaging, satellite imagery analysis, and quality control.
- Identify specific use cases where accurate and real-time object detection on images can provide significant value.

METHODOLOGY

The methodology for this project involved a systematic approach to implementing object detection using the YOLOv7 (You Only Look Once version 7) model on static images. The final objective was to achieve accurate and efficient object detection using YOLOv7, applied specifically to static image data.

1. Research and Understanding:

- Conducted comprehensive research on the YOLO (You Only Look Once) object detection model, specifically focusing on YOLOv7.
- Studied the principles and working mechanisms of YOLOv7 for accurate and efficient object detection on static images.

2. Implementation of YOLOv7:

- Implemented YOLOv7 for object detection, utilizing its efficient architecture and pre-trained weights from the COCO (Common Objects in Context) dataset.
- Integrated YOLOv7 into the system for detecting objects in static images, ensuring accuracy and efficiency in detection.

3. Evaluation and Fine-Tuning:

- Evaluated the performance of the YOLOv7 model on benchmark image datasets to assess detection accuracy and other performance metrics.
- Fine-tuned the YOLOv7 model and parameters to optimize detection performance on specific object categories or challenging visual contexts.

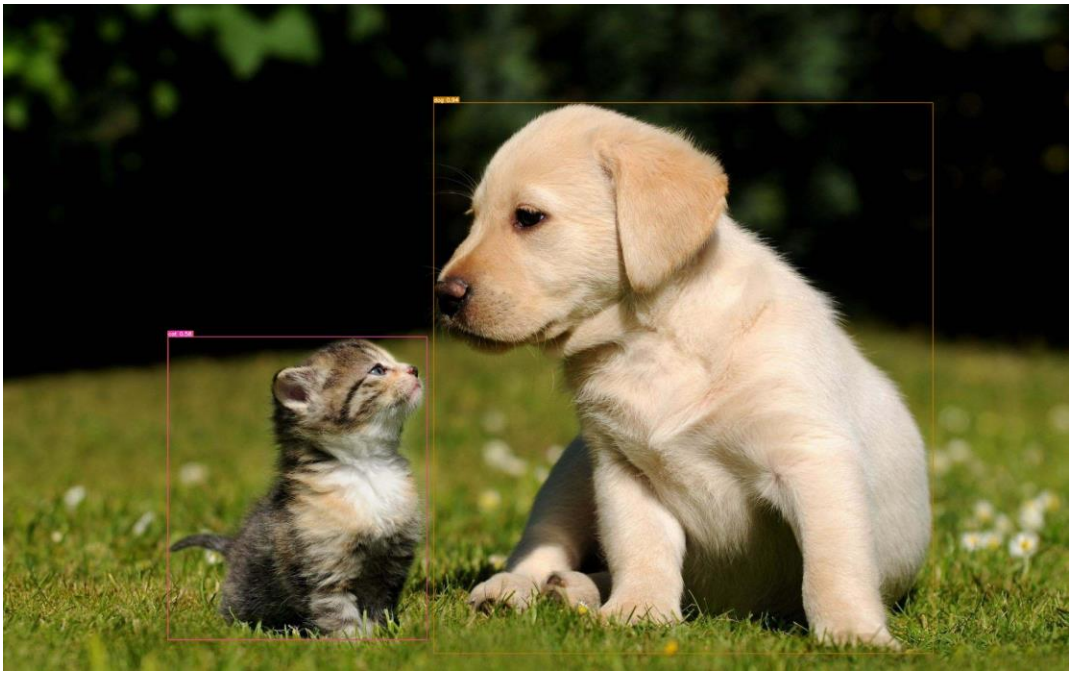
4. Input Image Format Compatibility:

- Ensured compatibility with various image formats for processing static images, allowing flexibility in input data sources.
- Validated the system's performance on static images, analyzing results and making necessary adjustments for accuracy.

RESULTS & DISCUSSIONS

Object Detection Test Input:

Object Detection Test Output:



CONCLUSION

In conclusion, the successful integration of YOLOv7 into our project for object detection marks a significant milestone in advancing defence-related technologies. This project showcases the efficacy of YOLOv7 in accurately identifying objects of interest within images, a capability crucial for defence operations.

The robust performance of YOLOv7 ensures high accuracy and precision in object detection, which is imperative for minimizing false alarms and enabling swift and accurate target identification. The project demonstrates the capability of YOLOv7 to operate effectively under varying conditions, including challenging environments with occlusions or fluctuating lighting.

The scalability and adaptability of YOLOv7 make it suitable for deployment across a range of defence applications, offering flexibility in addressing diverse operational needs. From static surveillance systems to mobile platforms, YOLOv7 proves its utility in enhancing situational awareness and aiding defence personnel in making informed decisions.

Moving forward, continued research and development in object detection technologies, particularly leveraging advancements in models like YOLOv7, will further strengthen defence capabilities. By improving detection and monitoring capabilities, these technologies contribute significantly to national security efforts, ensuring readiness to address evolving threats effectively.

APPENDIX – A (Source Code)

Object Detection using Yolov7

Training Yolov7

```
!python3 train.py --workers 5 --device 0 --batch-size 16 --data data/custom_data.yaml --img 512 512 --cfg cfg/training/yolov7_cust.yaml --weights '' --name yolov7 --hyp data/hyp.scra
```

```
2024-06-08 12:18:43.235402: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one
2024-06-08 12:18:43.235461: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one
2024-06-08 12:18:43.236796: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
2024-06-08 12:18:43.243468: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operati
To enable the following instructions: AVX2 AVX512F FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-06-08 12:18:44.295817: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
YOLOR 🚀 2024-6-8 torch 2.3.0+cu121 CUDA:0 (Tesla T4, 15102.0625MB)

Namespace(weights='', cfg='cfg/training/yolov7_cust.yaml', data='data/custom_data.yaml', hyp='data/hyp.scratch.p5.yaml', epochs=50, batch_size=16, img_size=[512, 512], rect=False, re
tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.3, cls_pw=1.0, obj=0.7, obj_pw=1
wandb: Install Weights & Biases for YOLOR logging with 'pip install wandb' (recommended)

                 from  n    params  module                                  arguments
  0                -1  1       928  models.common.Conv                      [3, 32, 3, 1]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1      8320  models.common.Conv                      [128, 64, 1, 1]
  5                -2  1      8320  models.common.Conv                      [128, 64, 1, 1]
  6                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  7                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  8                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  9                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
 10  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
```

```
     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     45/49     8.54G   0.06003   0.01293    0.0215   0.09446        65       512: 100% 9/9 [00:50<00:00,  5.63s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.31it/s]
                 all          15          15         0.5      0.0666      0.0358     0.00398

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     46/49     8.54G   0.06157   0.01196   0.02129   0.09482        74       512: 100% 9/9 [00:46<00:00,  5.20s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.59it/s]
                 all          15          15        0.26      0.0667      0.0657      0.0133

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     47/49     8.54G   0.06269   0.01255   0.02113   0.09637        48       512: 100% 9/9 [00:38<00:00,  4.25s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.01it/s]
                 all          15          15     0.00913         0.2     0.00552     0.00178

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     48/49     8.54G   0.05989   0.01198   0.02141   0.09327        52       512: 100% 9/9 [00:45<00:00,  5.02s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.21it/s]
                 all          15          15     0.00675         0.2     0.00292    0.000453

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     49/49     8.54G   0.06183   0.01197   0.02165   0.09545        65       512: 100% 9/9 [00:40<00:00,  4.46s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.21it/s]
                 all          15          15      0.0173       0.333      0.0138     0.00367
                 cat          15           5      0.0292         0.6      0.0251     0.00797
                 dog          15           5           0           0           0           0
              rabbit          15           5      0.0227         0.4      0.0163     0.00304
50 epochs completed in 0.708 hours.

Optimizer stripped from runs/train/yolov72/weights/last.pt, 74.8MB
Optimizer stripped from runs/train/yolov72/weights/best.pt, 74.8MB
```

Testing Yolov7

```
!python3 test.py --data data/custom_data.yaml --img 512 --batch 16 --conf 0.001 --iou 0.65 --device 0 --weights yolov7.pt --name yolov7_640_val

Namespace(weights=['yolov7.pt'], data='data/custom_data.yaml', batch_size=16, img_size=512, conf_thres=0.001, iou_thres=0.65, task='val', device='0', single_cls=False, augment=False
YOLOR 🚀 2024-6-8 torch 2.3.0+cu121 CUDA:0 (Tesla T4, 15102.0625MB)

Fusing layers...
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
Model Summary: 306 layers, 36905341 parameters, 36905341 gradients
 Convert model to Traced-model...
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:810: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor is being accessed. Its .grad attribute
  param_grad = param.grad
 traced_script_module saved!
 model is traced!

/usr/local/lib/python3.10/dist-packages/torch/functional.py:512: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered i
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
val: Scanning 'data/val/labels.cache' images and labels... 15 found, 0 missing, 0 empty, 0 corrupted: 100% 15/15 [00:00<?, ?it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95:   0% 0/1 [00:00<?, ?it/s]Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=t
Skipping out-of-bounds index: gc=1, detection_classes[m1[j]]=tensor([16], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=0, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=1, detection_classes[m1[j]]=tensor([16], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=1, detection_classes[m1[j]]=tensor([16], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=1, detection_classes[m1[j]]=tensor([16], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=0, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=0, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=1, detection_classes[m1[j]]=tensor([16], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=0, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=0, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)

Skipping out-of-bounds index: gc=0, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=0, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
Skipping out-of-bounds index: gc=2, detection_classes[m1[j]]=tensor([15], device='cuda:0', dtype=torch.int32)
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:06<00:00,  6.57s/it]
                 all          15           0           0           0           0           0
Speed: 65.3/157.7/223.0 ms inference/NMS/total per 512x512 image at batch-size 16
/usr/local/lib/python3.10/dist-packages/seaborn/matrix.py:202: RuntimeWarning: All-NaN slice encountered
  vmin = np.nanmin(calc_data)
/usr/local/lib/python3.10/dist-packages/seaborn/matrix.py:207: RuntimeWarning: All-NaN slice encountered
  vmax = np.nanmax(calc_data)
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:810: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor is being accessed. Its .grad attribute
  param_grad = param.grad
Results saved to runs/test/yolov7_640_val
```

Detecting Objects in an Image

```
!python3 detect.py --weights yolov7.pt --conf 0.25 --img-size 640 --source picture.jpg

Namespace(weights=['yolov7.pt'], source='picture.jpg', img_size=640, conf_thres=0.25, iou_thres=0.45, device='', view_img=False, save_txt=False, save_conf=False, nosave=False, classe
YOLOR 🚀 2024-6-8 torch 2.3.0+cu121 CUDA:0 (Tesla T4, 15102.0625MB)

Fusing layers...
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
Model Summary: 306 layers, 36905341 parameters, 6652669 gradients
 Convert model to Traced-model...
 traced_script_module saved!
 model is traced!

/usr/local/lib/python3.10/dist-packages/torch/functional.py:512: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered in
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
1 cat, 1 dog, Done. (16.6ms) Inference, (614.8ms) NMS
 The image with the result is saved in: runs/detect/exp3/picture.jpg
Done. (1.139s)
```

REFERENCES

[1] Papageorgiou, C., & Poggio, T. (2000). A trainable system for object detection. International Journal of Computer Vision, 38, 15-33.

[2] Diwan, T., Anirudh, G., & Tembhurne, J. V. (2023). Object detection using YOLO: Challenges, architectural successors, datasets, and applications. Multimedia Tools and Applications, 82(6), 9243-9275.

[3] Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2022). A review of YOLO algorithm developments. Procedia Computer Science, 199, 1066-1073.

[4] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2021). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[5] Wang, C., Bochkovskiy, A., & Liao, H. Y. M. (2021). YOLOv5: 1.5x Faster on CPU, 3x Faster on GPU. arXiv preprint arXiv:2104.02126.